Software Risk Triage
zeroing in on ones problems

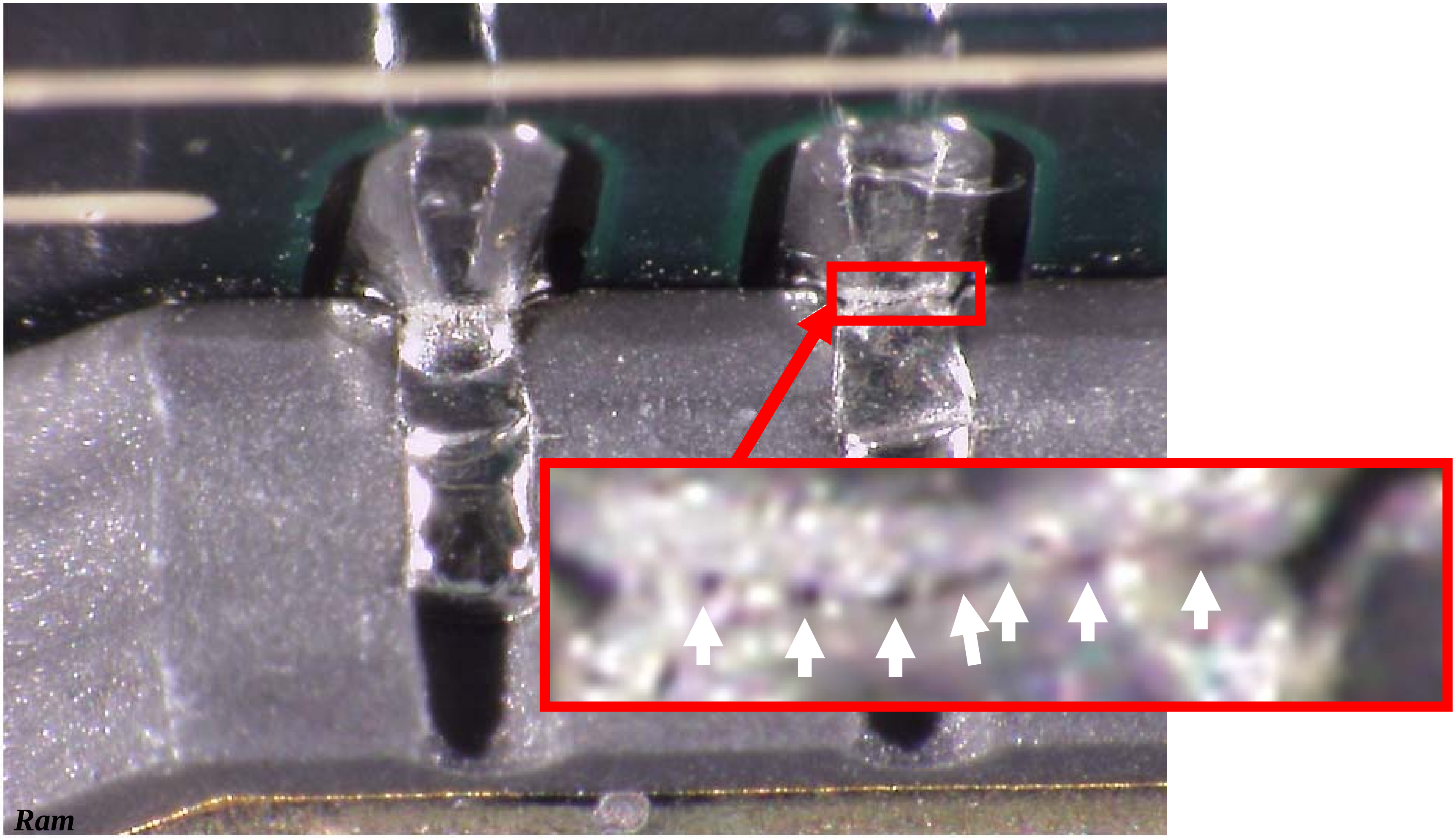
Martin S. Feather

Jet Propulsion Laboratory
California Institute of Technology

My research has been carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration and funded through the NASA ESMD Integrated Modeling and Simulation effort, NASA's Office of Safety and Mission Assurance, NASA's former Codes R & T, and JPL's internal Research and Technology Development program.

This presentation draws upon many peoples' material (but the mistakes are mine).

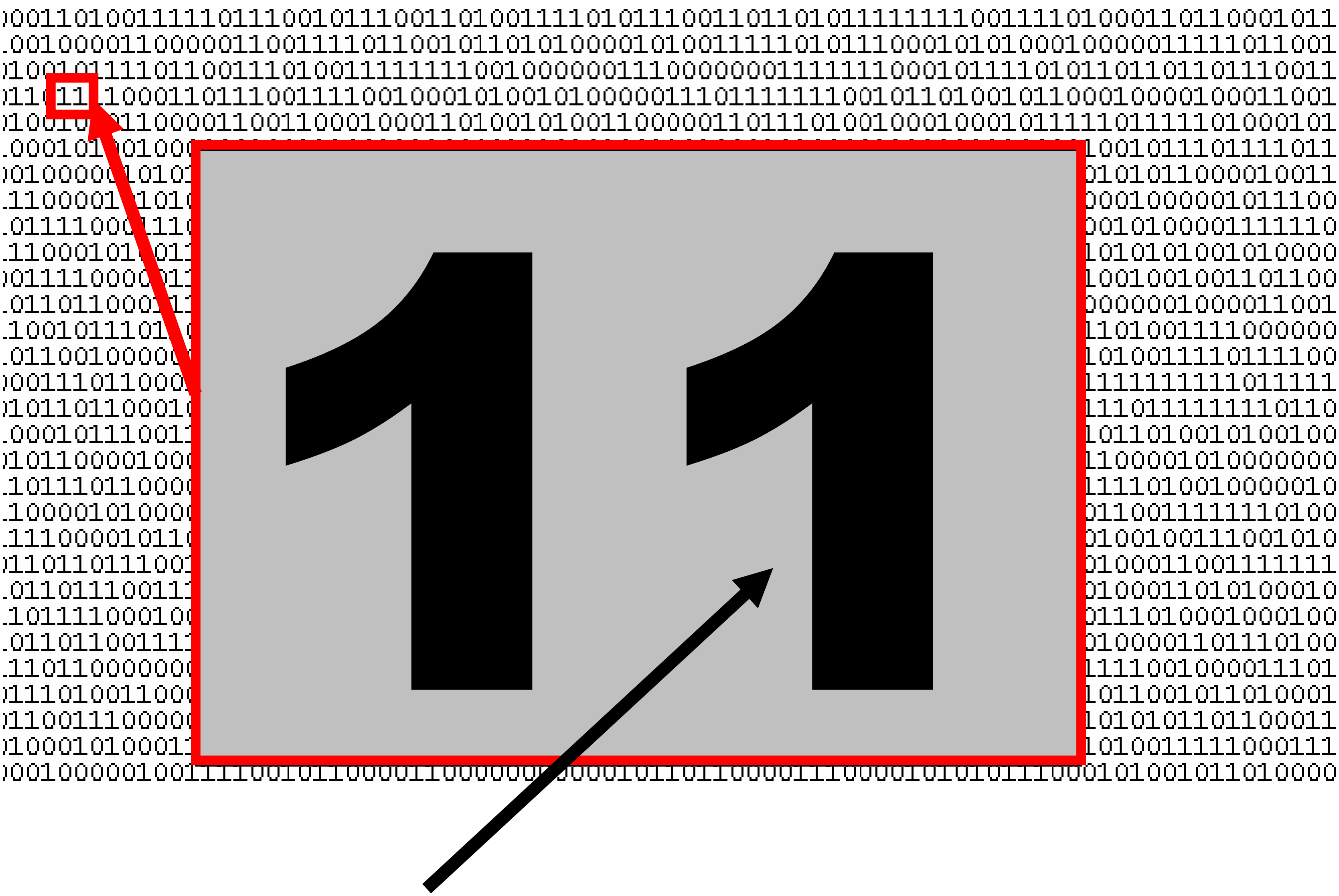
Hardware faults



Optical photograph of a cracked interconnect

Photo courtesy of Rajeshuni Ramesham, JPL

Software faults can be hard to spot



A faulty “1” (should be a “0”)

Worse yet…

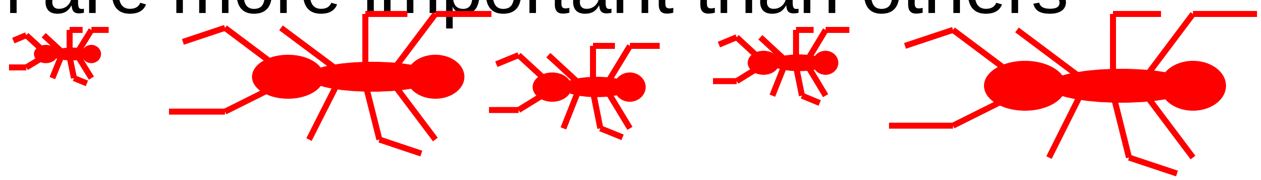
- By the time they are obvious, it's often very expensive to correct them:
  net result:
  crisis management, not ***risk*** management
  expensive, rushed, far from perfect

- There are very many kinds of software faults ("bugs" / "defects")
  I have seen a breakdown of 200+ *kinds* of software defects!
  I have seen a software risk checklist of 400+ entries!

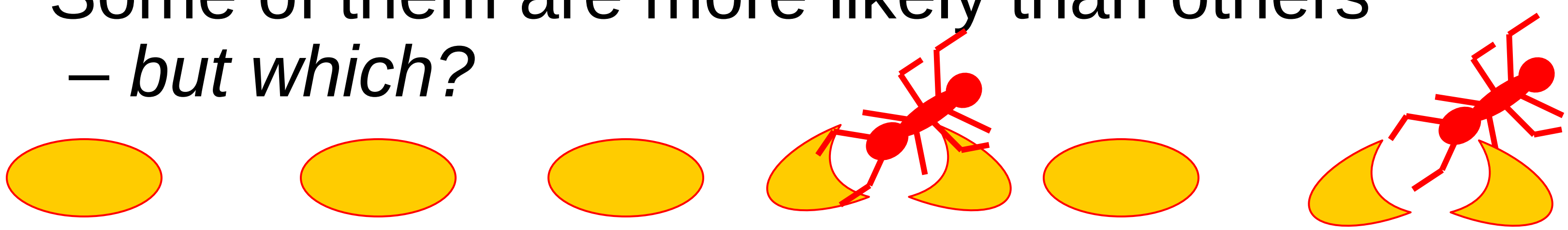
What’s needed

Need to zero in and assess critical software problems, so know which to address

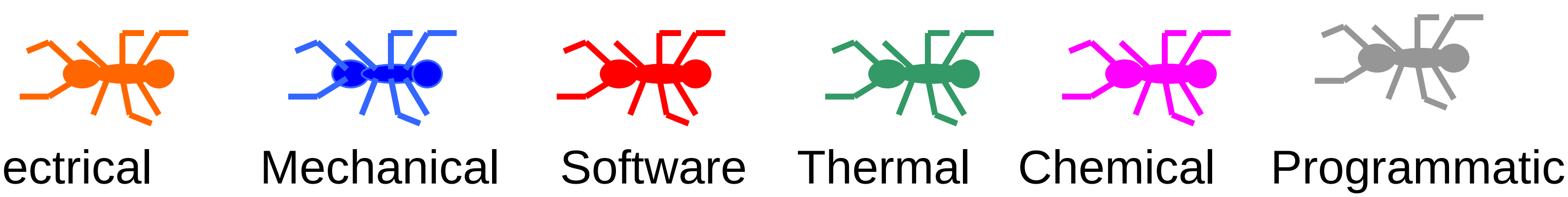
- Some of them are more important than others
  – *but which?*



- Some of them are more likely than others
  – *but which?*



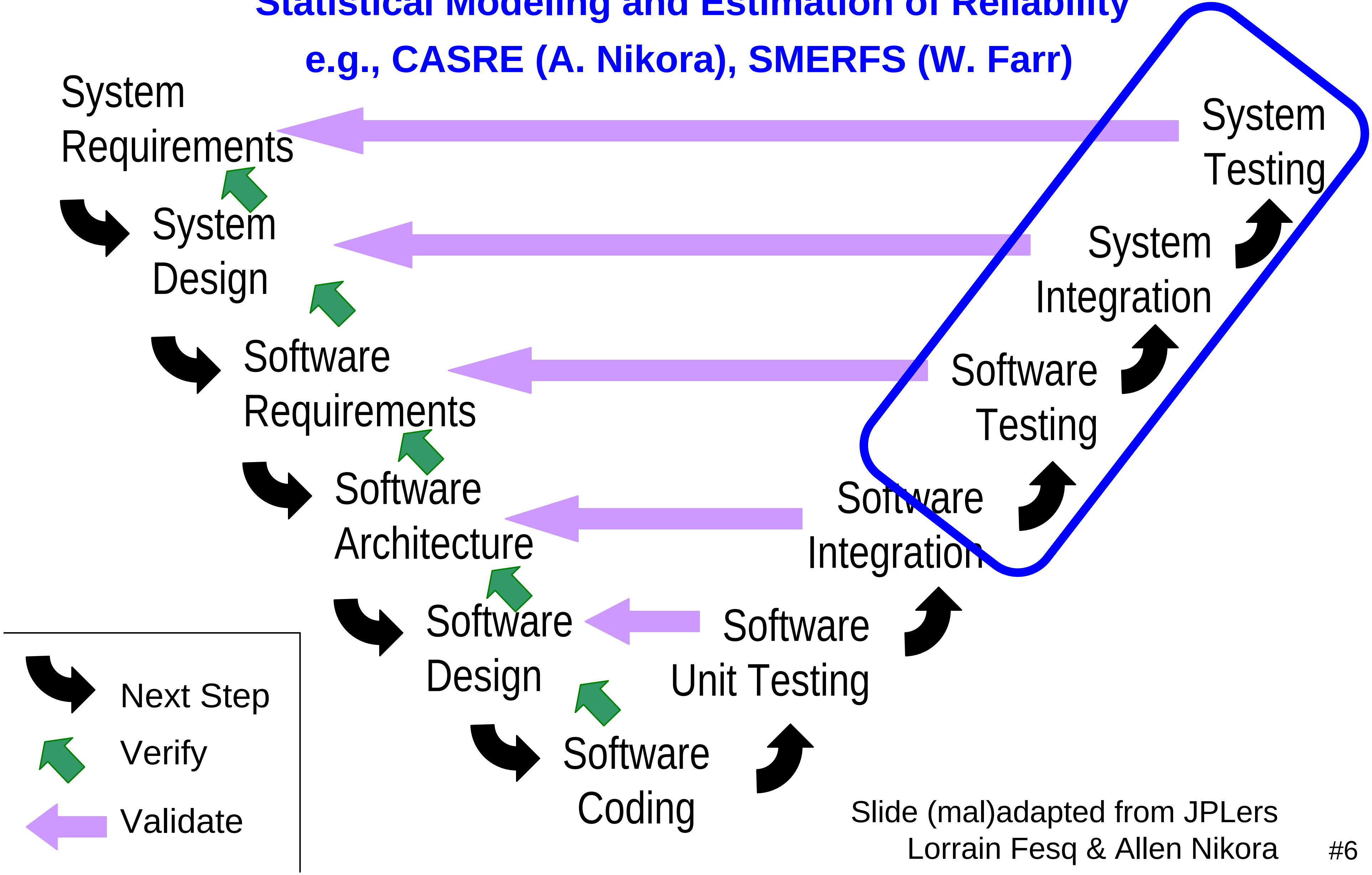
- *How do they compare to other risks of the project?*



Software <u>Reliability Estimation</u>

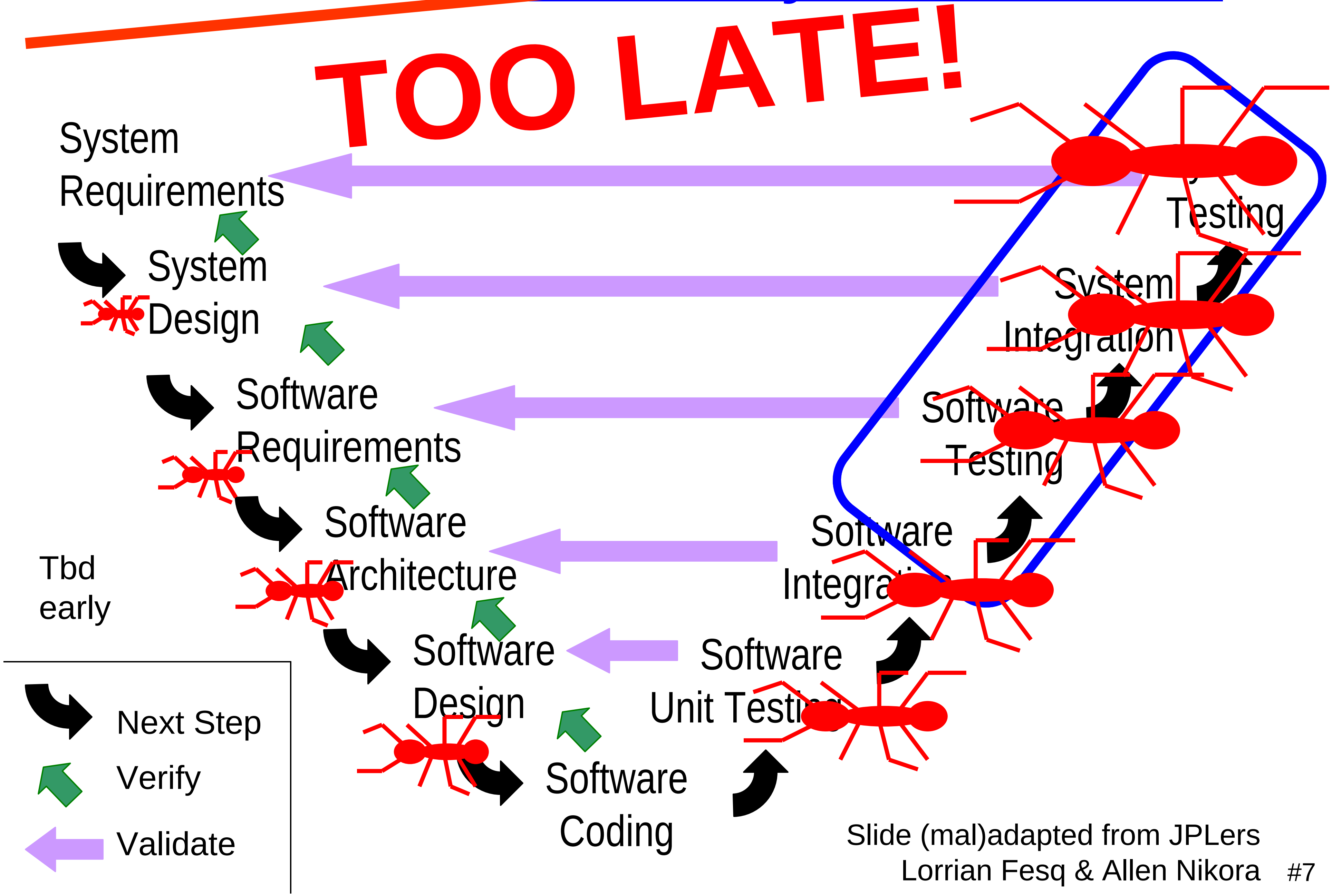
Statistical Modeling and Estimation of Reliability

e.g., CASRE (A. Nikora), SMERFS (W. Farr)



Slide (mal)adapted from JPLers
Lorrain Fesq & Allen Nikora

~~Software Reliability Estimation~~

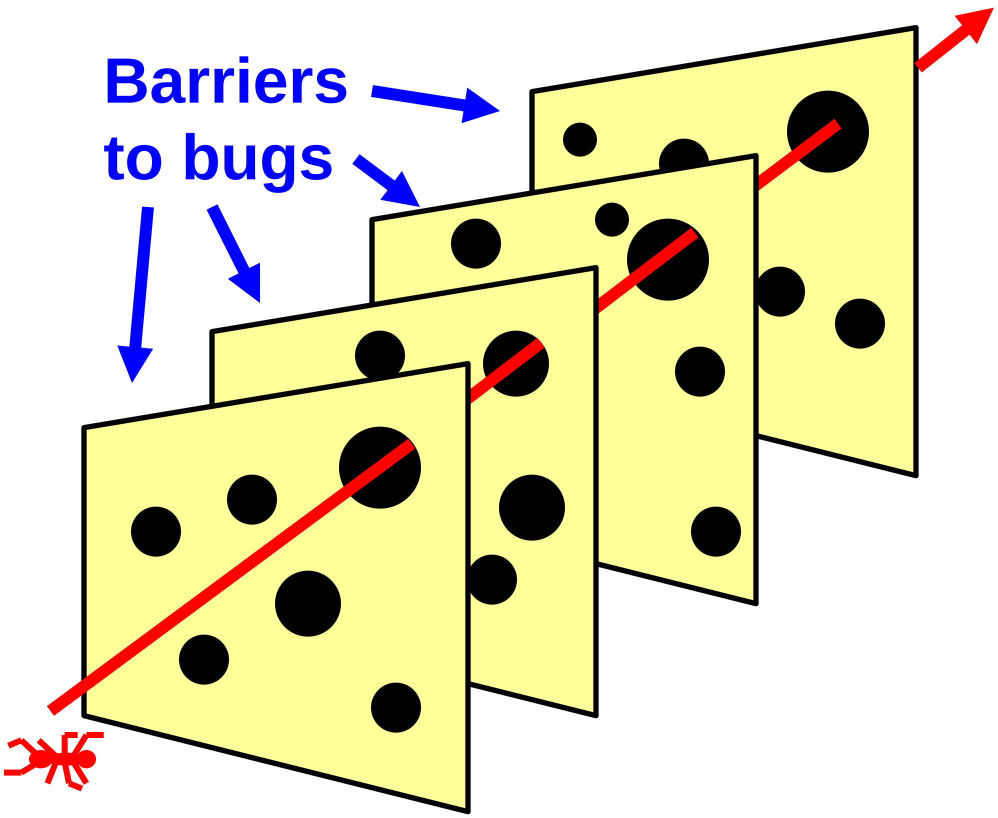


How can a

Risk Perspective

help?

James Reason's "Swiss Cheese Model"

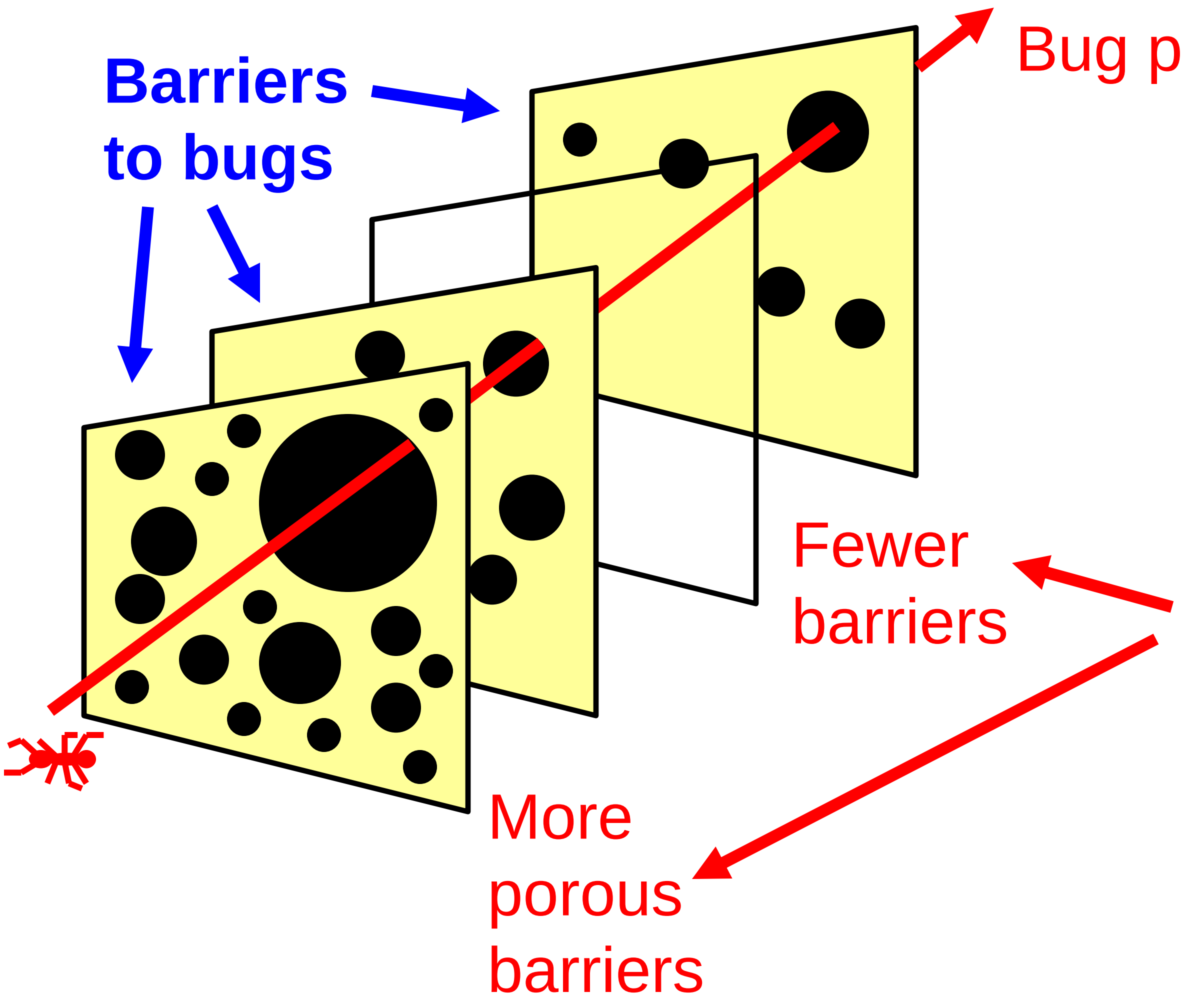


Barriers have holes (they are *imperfect*)

To decrease risk, can use *better* barriers (fewer/smaller holes) and/or *more* barriers

– but at a cost (time, $, personnel, …)

Faster, Better Cheaper in the "Swiss Cheese Model"



Faster and cheaper, but *riskier*

- yes, but by *how much?*

Mars Polar Lander testing

… probable cause of the loss of MPL … premature shutdown of the descent engines, resulting from a vulnerability of the software to transient signals … the leg deployment test was not repeated after the wiring error was corrected. A rerun of that test … might have detected the software logic problem …

… software was not tested…in the flight configuration…

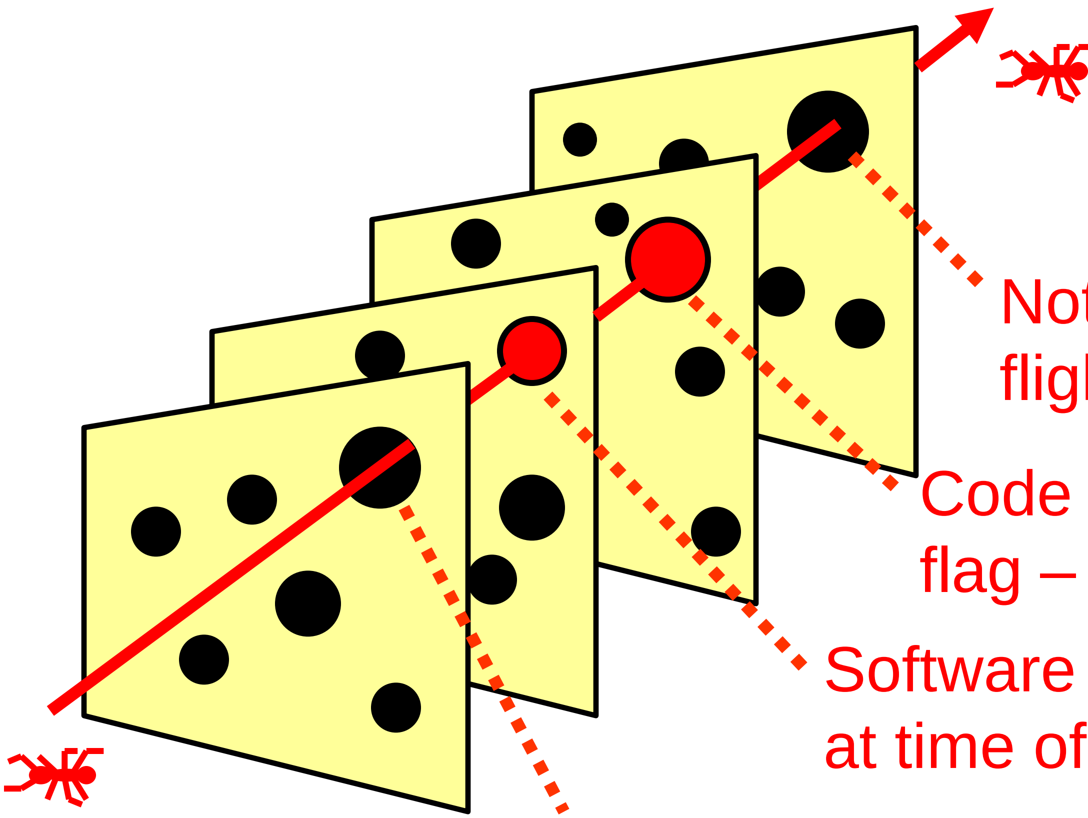
From "Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions"

…86.7% to 99.96% chance [that 3 legs' deployment be interpreted as a touchdown signal]…

From "Low-Cost, Light-Weight Mars Landing System", R. Warwick, IEEE Aerospace Conf. 2003

So the chance that this <u>would</u> have occurred on landing but <u>not</u> on a rerun of the post-correction test is 0.04% - 11.5%

MPL & James Reason's "Swiss Cheese Model"



n.b., in other respects these were risk *mitigators*!

Not tested in flight configuration

Code changed to add touchdown flag – new version had the bug

Software unnecessarily running at time of leg deployment

Faulty requirements flowdown (system to software)

Good news: takes a combination of flaws to lead to failure – the individual flaws are unlikely, so their combination is even less so.
Bad news: there are lots and lots of failure arrows.

Mars Climate Orbiter

Root Cause:
Failure to use metric units in the coding of a ground software file, "Small Forces," used in trajectory models

Contributing Causes:

1. Undetected mismodeling of spacecraft velocity changes

2. Navigation Team unfamiliar with spacecraft

3. Trajectory correction maneuver number 5 not performed

4. System engineering process did not adequately address transition from development to operations

5. Inadequate communications between project elements

6. Inadequate operations Navigation Team staffing

7. Inadequate training

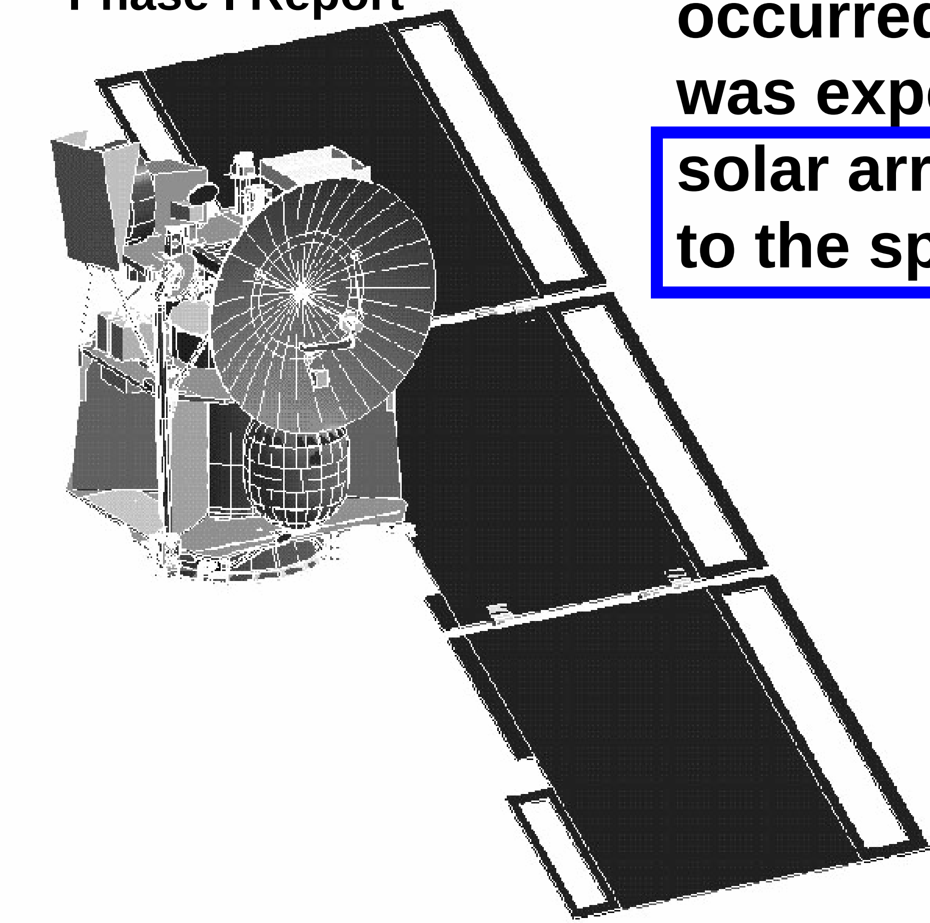
8. Verification and validation process did not adequately address ground software

From "Mars Climate Orbiter Mishap Investigation Board Phase I Report"

Mars Climate Orbiter

From "Mars Climate Orbiter Mishap Investigation Board Phase I Report"

… propulsion maneuvers … to remove angular momentum buildup … occurred 10-14 times more often than was expected … because the MCO solar array was asymmetrical relative to the spacecraft body



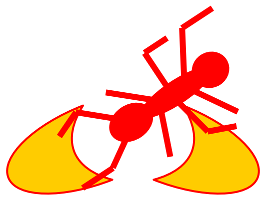
A design decision that increased the number of maneuvers, <u>each of which utilized software</u>. Unfortunately, the errors were cumulative.

How to gauge software risk *this early* in development???

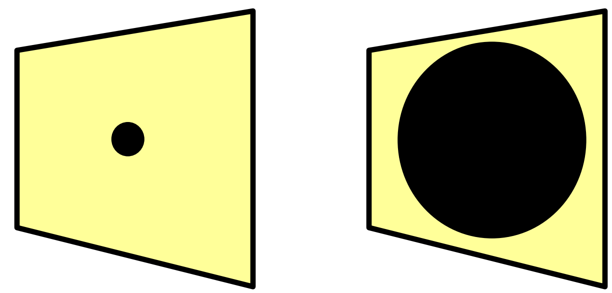
Mars Climate Orbiter In Cruise Configuration
(from http://mars.jpl.nasa.gov/msp98/orbiter/cruise.html)

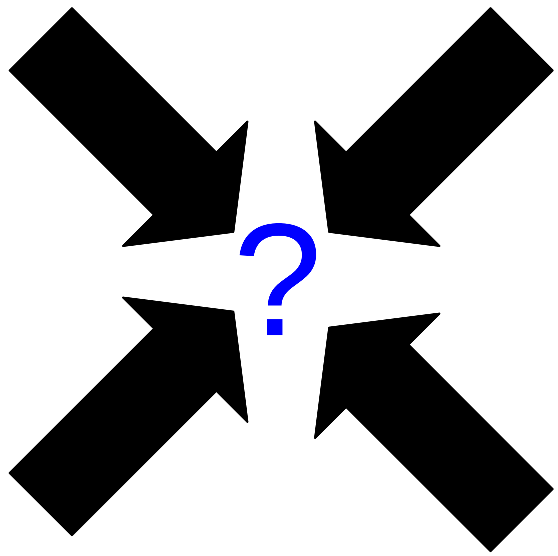
A Unified Risk Model for Software?

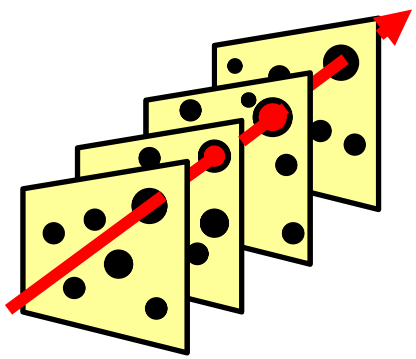
How prevalent are software defects?



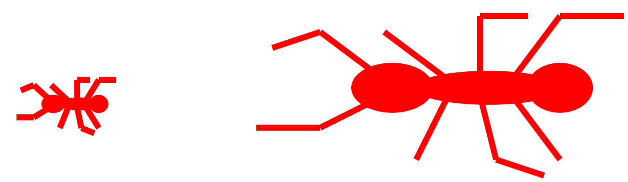
How effective are defect prevention & detection measures?







How do defects propagate to become software failures?



How would software failures affect the mission?

Prevalence and effectiveness

Needed: data on defect prevalence, and effectiveness of defect preventions & detections

Software community wide efforts in this direction: e.g., "What We Have Learned About Fighting Defects" http://www.CeBASE.org

… Better collection and sharing of metrics

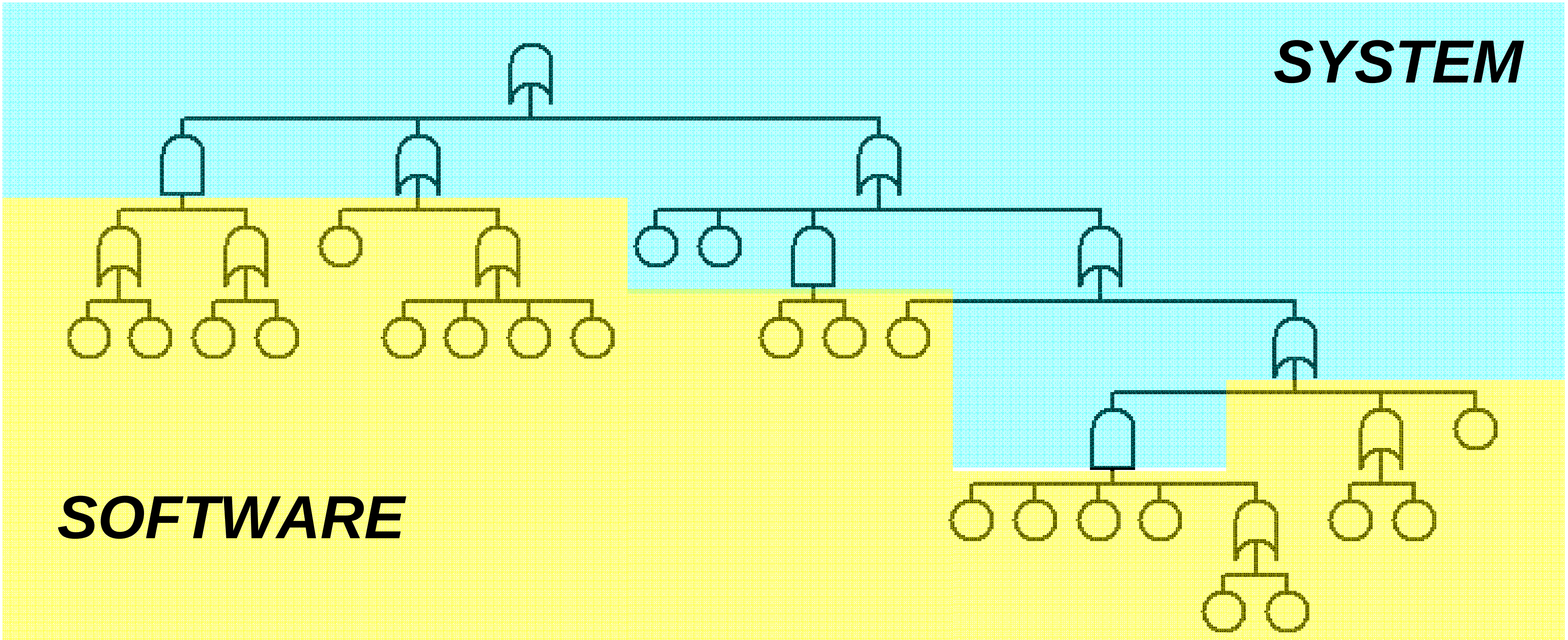
… More experiments (testbeds, benchmark problems, …)

Use Software Reliability Estimation to calibrate and validate software defect models

Software/System interface

How would software failures affect the mission?

Use fault trees that straddle the SOFTWARE / SYSTEM interface – e.g., work by R. Lutz, JPL; J. Dugan, U. Virginia.

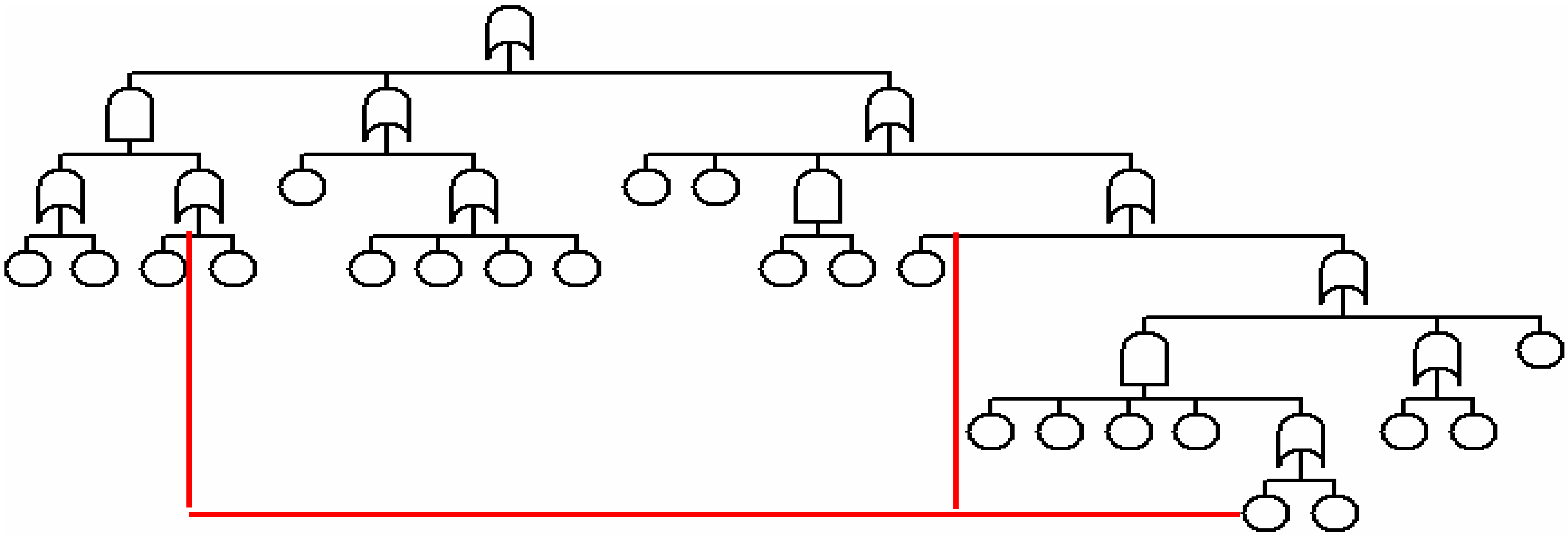


Gives insight into the where to focus

Software fault propagation

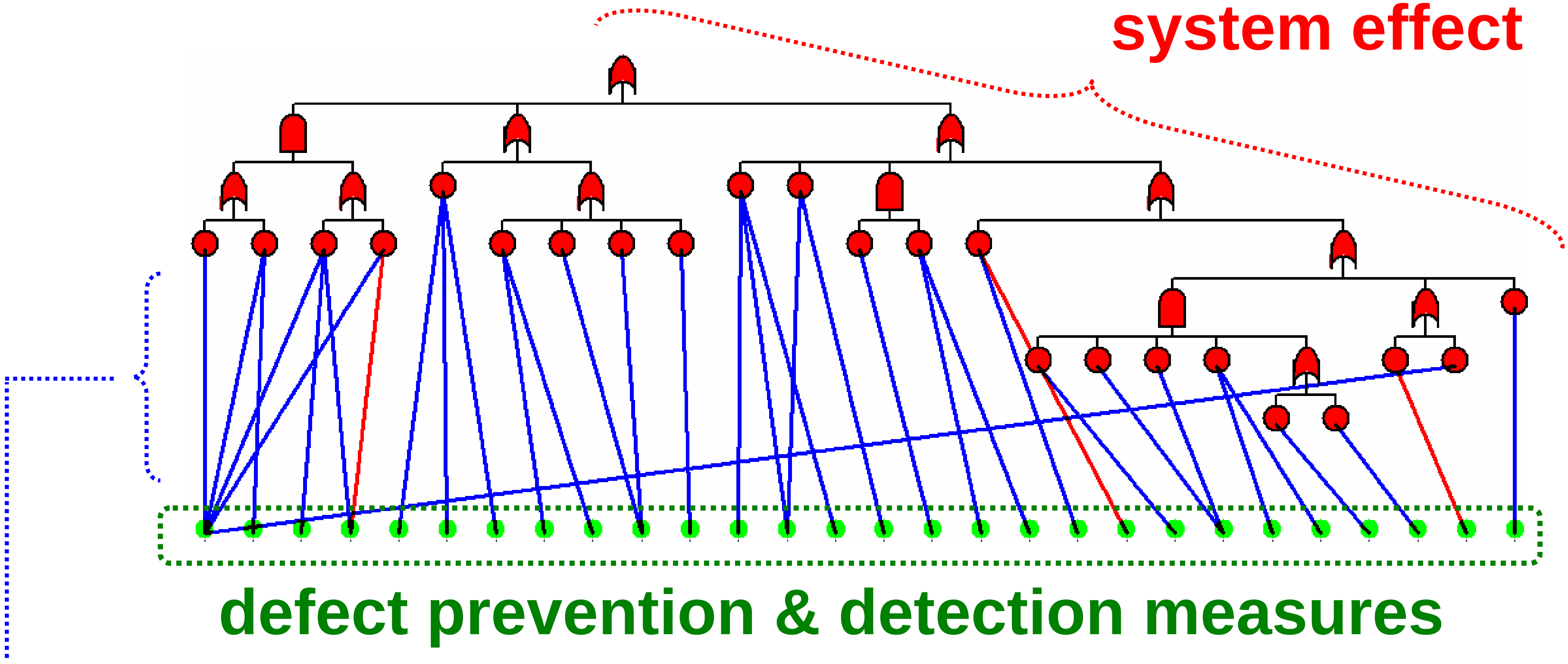
How do defects propagate to become software failures?

Can fault trees be used *within* the software architecture itself? E.g., current studies by C. Smitds, UMD



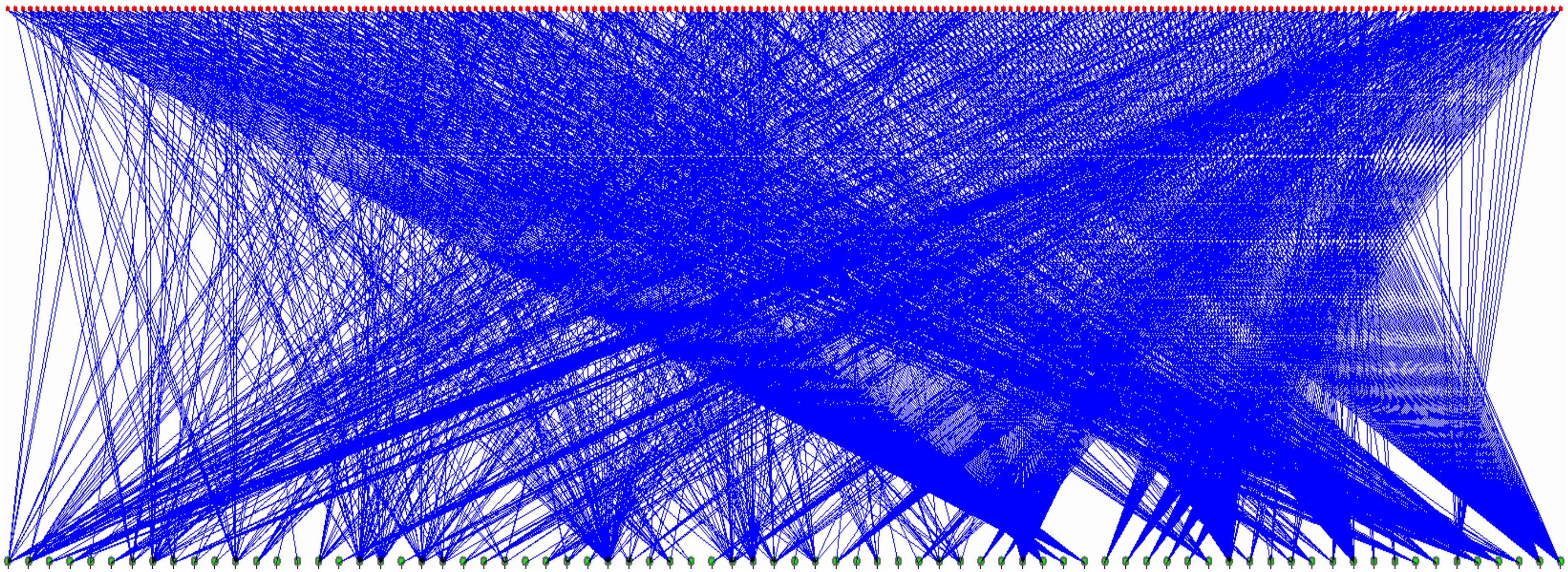
Challenge: potentially many non-local effects of software…

Tying it all together!



Connections between the prevention & detection measures and the faults (software defects) they decrease (or, on some occasions, increase)

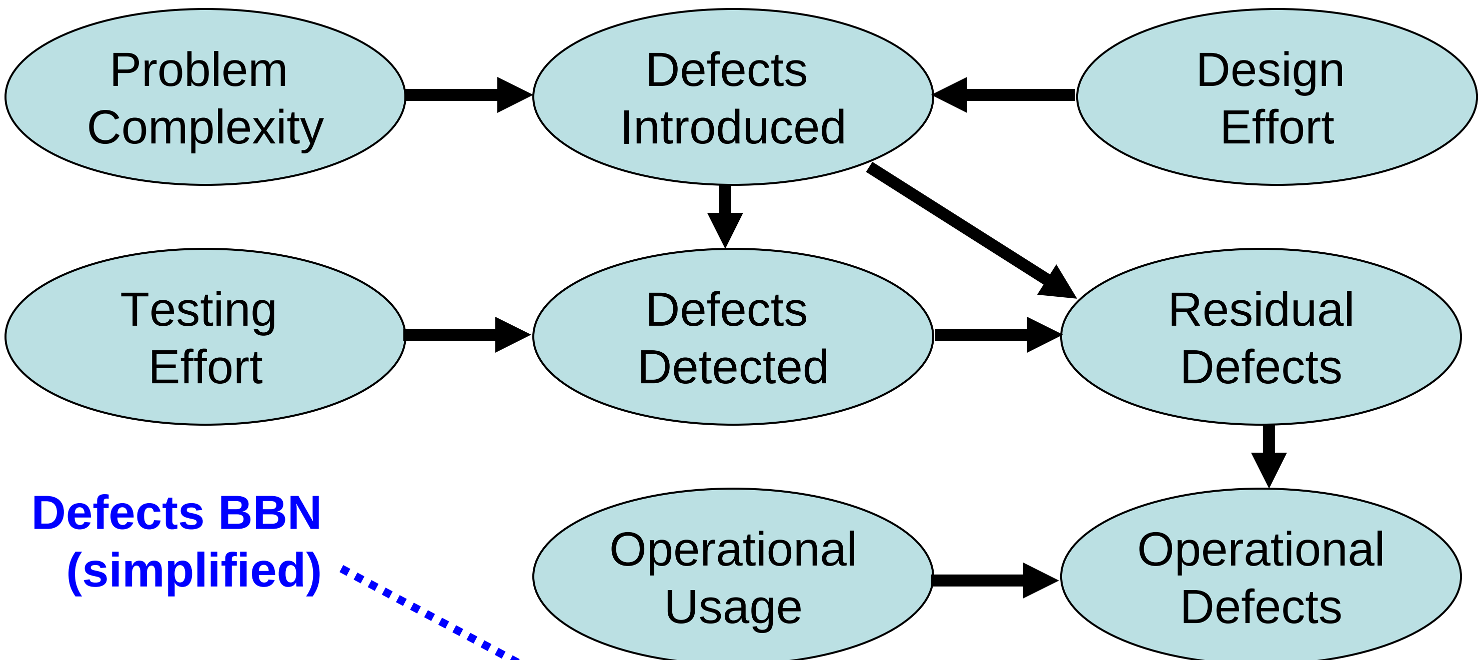
Note: Reality may be complex…



Connections between 210 software defects and 76 defect prevention & detection measures !

Tying it all together! Bayesian models?

Accommodate both expert judgment, and data as it becomes available; capture cause-and-effect relationships; can mix discrete (e.g., low/med/high) and numeric values.



Defects BBN (simplified)

Current studies by N. Fenton et al (Univ. London), and by J. Dugan et al (Univ. Virginia)

I believe a Risk Perspective <u>can</u> help!

While many risk techniques can’t be used “as is” on software, it is not a forlorn hope to expect that they can be adapted for software.

This will take some effort by, and cooperation among, the risk and software communities.

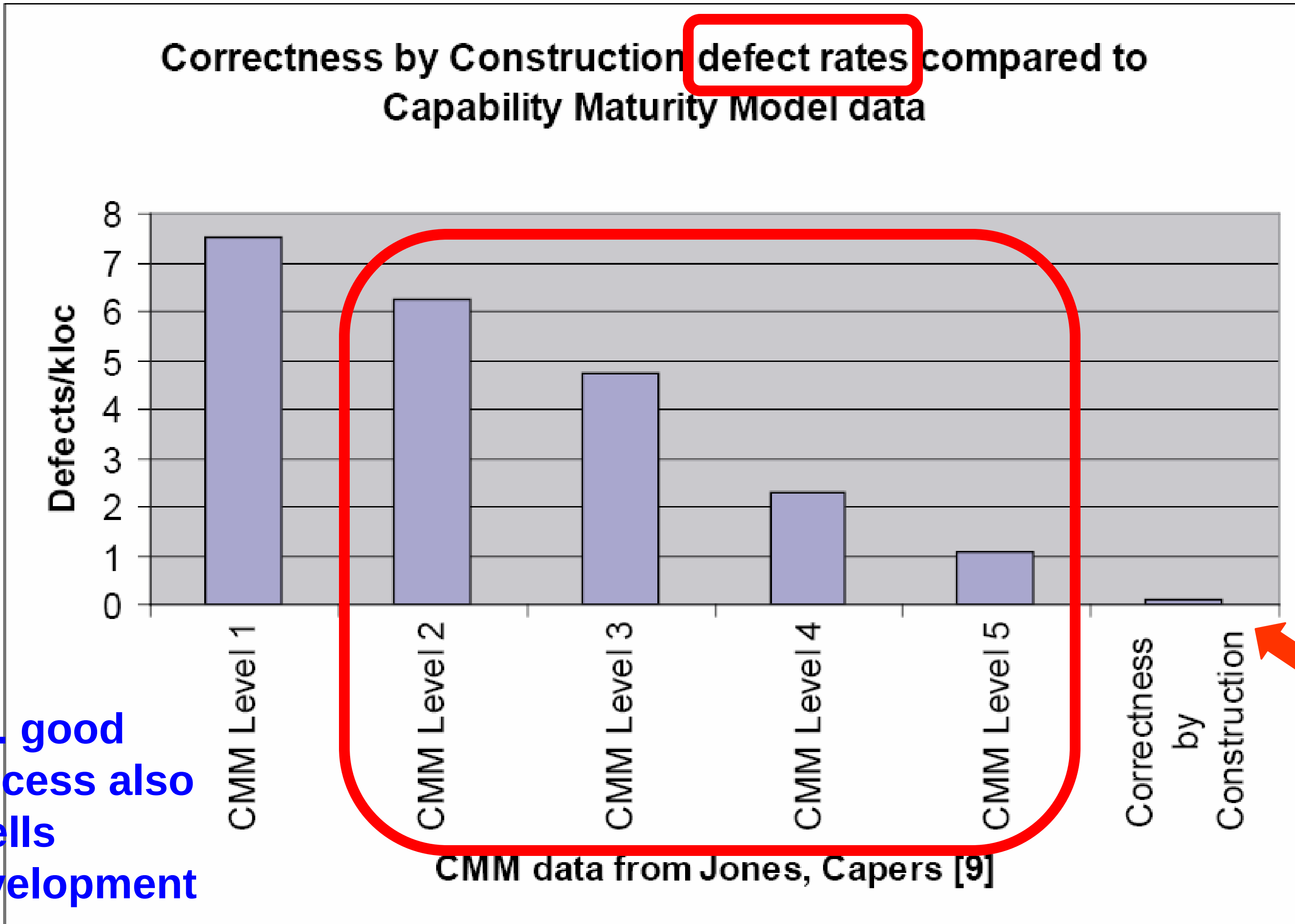
The need is there! The time is ripe!

Backup Slides

From the 2005 Florida Statues:

316.1985 Limitations on backing.--
(1) The driver of a vehicle shall not back the same unless such movement can be made with safety and without interfering with other traffic.
…

Good Process – good, but not enough



Each an order of magnitude reduction

0.1 defects/kloc

Wow! When does this work?

n.b. good process also quells development risks

Extracted from
"**The Challenge of Low Defect, Secure Software – too difficult and too expensive?** "
Martin Croxford, in *The DoD SoftwareTech*, July 2005 (http://iac.dtic.mil/dacs)

Software is indispensable ☺

Software indispensable – think information processing…

The good news: software often involved in **mitigating risk**

Mitigates development-time risk:

- Mediates between known disparities of hardware
- Accommodates unplanned discrepancies

Mitigates operations-time risk:

- Plays an active role in fault protection / fault detection, isolation & recovery
- Allows for deployment and change post-launch e.g., Galileo's high-gain antenna, and later radiation damage of A2D in a gyro; DS1's star tracker failure during extended mission (used camera instead)

Software is problematic ☹

Software indispensable – think information processing…

The bad news: software often **a risk contributor**

Contributes development-time risk:

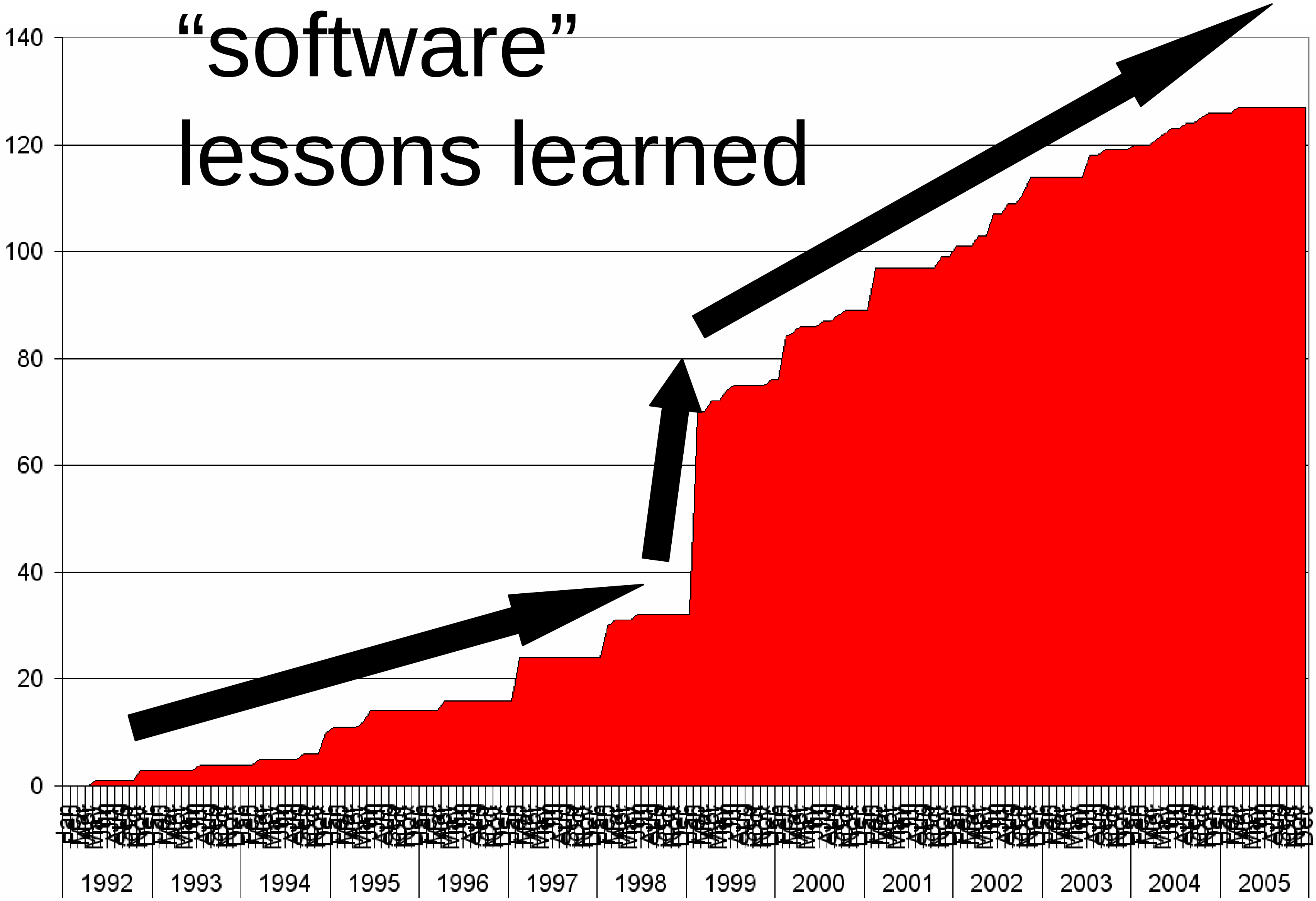
Overbudget, overschedule

Contributes operations-time risk:

Numerous instances of problems, ranging from loss of some science data to loss of entire mission
(I'm sure you can think of examples…)

More bad news: the problem continues…

Cumulative number of “software” lessons learned



Data source: http://nen.nasa.gov/portal/site/llis Topic = Software 11/22/2005